

COMPUTER CENTRE BULLETIN

*Volume 2, Number 1.
13th January, 1969.*

*Editor:
H. L. Smythe.*

EDITORIAL COMMENT

This Bulletin is the first issue of Volume 2 for 1969. I would sincerely like to thank those who contributed articles and ideas towards Volume 1, and those who have shown interest and enthusiasm in the production of the Bulletin.

Last year, there was a considerable number of "teething" problems that delayed publication, and I apologise for these delays. Many of these difficulties have now been remedied, so that the production of the Bulletin should function more smoothly this year.

The Bulletin is distributed according to two separate mailing lists, one listing organisations and departments, the other, individuals. Readers will recall it was stated in the first edition of the Bulletin that the individual mailing list would be maintained on an annual basis. Since Volume 1 was so short, however, the existing individual mailing list will continue throughout 1969. Hence, it is not necessary to re-apply for inclusion in the current mailing list. It would also be appreciated if any persons not wishing to receive further copies would notify the Editor as soon as possible. An application form for new subscribers can be found on the last page of this issue.

Articles in this issue include an improved method of using variable dimensions in FORTRAN IV, discuss some problems in the use of the Permuted Index Package, and introduce the list processing language WISP. A further article on the Computer Centre staff introduces the Programmers and the Secretary.

STAFF OF THE COMPUTER CENTRE

INTRODUCING THE PROGRAMMERS

Currently, the Computer Centre has on its staff, three full-time Programmers, John Williams, Bill Fulton, and John Row. Their main task is *systems programming*. This involves the writing of useful utility packages and subroutines, the correction of errors in compilers, assemblers, and program packages, and ensures that clients are able to make effective use

of the available software. They are presently engaged in the programming of a new operating system for the PDP 10. This will enable the computer, which is basically capable of handling only one job at a time, to give service to a number of computing tasks. These may be from remote terminals or from the batch input device, i.e. the card reader, so that all jobs appear to run concurrently.

The Programmers also give programming courses to the undergraduate students of some departments, to postgraduate students in the Diplomas of Automatic Computing and Information Processing, and to computer users, both internal and external. In addition, they assist in the supervision of programming projects assigned to the Diploma students, and are available, at specified times, for consultation with Computer Centre clients who are experiencing programming difficulties.

Both John Williams and Bill Fulton majored in mathematics at this University and hold postgraduate Diplomas in Automatic Computing. Their current research interests centre around the processing of non-numerical information on the computer. In this connection, they have jointly developed a list-processing compiler for the GE 225. John Row, who holds a degree in electrical engineering, completed the Diploma of Automatic Computing in 1968, and is taking up duty from 20th January, 1969.

INTRODUCING THE SECRETARY

The Secretary of the Computer Centre, Miss Delphine Dare, performs the combined secretarial duties of both the Computer Centre and of the Department of Computer Science. The Secretary is responsible for typing manuals and technical write-ups, and for organizing programming courses held throughout the year. In addition, the Secretary reproduces material for student use within the Department of Computer Science. General enquires and queries on courses held by the Department of Computer Science can be directed initially to the Secretary (ext. 688).

PROGRAMMING ADVICE

ANOTHER LOOK AT VARIABLE DIMENSIONS

By studying the example given below, it is possible to observe three

simple rules which greatly facilitate the use of variably-dimensioned arrays in FORTRAN.

Suppose there are three arrays A, B, and C, of sizes 4x5, 5x4, and 10x10, respectively. We wish to add together all those elements of A contained in the "rectangle" of size 3x2 situated in the "top left-hand corner" of A (see Figure 1). We treat similarly the 3x3 rectangle at the corresponding corner of B and the 2x3 rectangle in C.

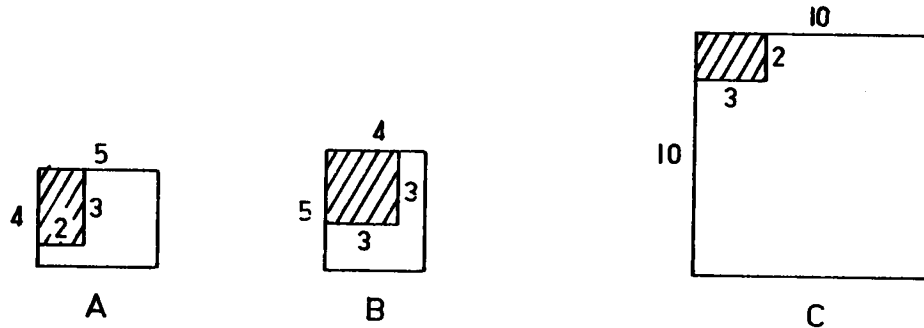


Figure 1

Preferring not to write out a piece of code three times when once is adequate, we then consider how a subroutine might be constructed to reduce our work. However, it is impossible to dimension the dummy array (P, say) in the subroutine to any fixed size, since P will stand for arrays of different sizes at different times. In FORTRAN IV, this difficulty can be overcome by giving P a *variable* size such as M x N:

```

SUBROUTINE ADD (P, M, N, K, L, SUM)
  DIMENSION P(M, N)
  SUM = 0.0
  DO 10 I = 1, K
    DO 10 J = 1, L
10    SUM = SUM + P(I, J)
  RETURN
END
```

A suitable main program would then be:

```
      DIMENSION A(47, 59), B(5, 4), C(10, 10)
      .....
      .....
      CALL ADD (A, 4, 5, 3, 2, ASUM)
      CALL ADD (B, 5, 4, 3, 3, BSUM)
      CALL ADD (C, 10, 10, 2, 3, CSUM)
      .....
      .....
      END
```

Note that, in the *main* program, each array has been given its actual *fixed* dimension. *It is not possible to give variable dimensions to an array in the main program.*

Next, observe that the name (P) and size (M, N) of the dummy array in the subroutine both appear in the subroutine's argument list. This is an example of the general rule that *the name and each dimension of any array that is assigned variable dimensions in a subprogram must be included in the argument list for the subroutine.**

Lastly, note that, in the example, the second and third arguments at each call to the subroutine, have been selected so that M and N in the subroutine stand for the actual physical size of the array declared in the main program, e.g. ⁴~~7~~ and ⁵~~9~~ for A. It is very important to distinguish this physical size from the size of the subarray which is referenced by the subroutine. Thus, the following code would *not* give the desired result:

```
      SUBROUTINE ADD (P, K, L, SUM)
      DIMENSION P(K, L)
      SUM = 0.0
      .....
      .....
      END
```

* In actual fact, because arrays are stored internally with the first subscript varying most rapidly, the dimension corresponding to the last subscript need not be included in the argument list.

This example illustrates a general rule:

In calls to a subprogram containing variably-dimensioned arrays, each actual argument corresponding to a dummy argument which occurs in the subprogram's DIMENSION statement must have the value of the actual physical dimension of the array as specified in the DIMENSION statement of the main program.

By following the above three rules, one has complete freedom to manipulate variably-dimensioned arrays in a main program as well as in subprograms. Note, however, that a small number of older library subprograms does not adhere to the third rule above. This can be seen by perusing their argument lists. When these latter subroutines are used, it is possible to refer to variably-dimensioned arrays only within subroutines in which the arrays are variably-dimensioned, and never within a main program. An attempt is being made to replace these unsatisfactory routines as quickly as possible.

DATA STATEMENT

The present version of the GE 225 FORTRAN IV Compiler will not accept the combination of characters "/" in the DATA statement i.e. do not use a statement such as:

```
DATA X, Y, Z /-4.0, 7.3, 11.6/
```

It may be possible to overcome this problem by re-ordering the variables so that the first is positive. In the above example, it is possible, e.g.

```
DATA Y, X, Z/7.3, -4.0, 11.6/
```

LOOP INDEX

The present version of the GE 225 FORTRAN IV Compiler will not allow a variable that is in COMMON to be used as an index in an implied DO loop. (see Example 1 (a)). The use of an argument of a subprogram as an index within the subprogram is also illegal (see Example 1 (b)).

(a)	(b)
COMMON I	SUBROUTINE SBR(A, N)
.
.
.
READ 10, (A(I), I = 1, 10)	PRINT 20, (A(N), N = 1, M)
.
.
.

Example 1. Illegal Loop Index

LIBRARY PROGRAMS

NEW PROGRAMS

DETERM - Matrix Determinant (D4.201)

This FORTRAN IV subroutine has been converted from CARD FORTRAN (DETERMINANT - D4.200), but the calling sequence has been slightly altered to conform with the variable dimensioning technique as described previously. It calculates the determinant of an n by n matrix.

RGENMT - Generate Real Symmetric Matrix (D4.271)

This subroutine is written in FORTRAN IV. It generates a real symmetric matrix with known determinant, and providing a certain condition holds, all elements of the inverse are integers. The values of the elements of the inverse can be calculated independently of the inversion process.

EIGN - Eigenvalues and Vectors of a Real Symmetric Matrix (D4.272)

This FORTRAN IV subroutine computes all the eigenvalues and eigenvectors of a real symmetric matrix. The eigenvectors are orthogonal even when the matrix is derogatory (has repeated eigenvalues). The matrix may be reduced to triple diagonal form by Householder's method, after which the QR transformation may be applied to reduce the matrix to diagonal form.

RECENT PUBLICATIONS

The following publications are available at the Computer Centre:

TECHNICAL MEMORANDUM NO. 5: *Digital-Analogue Simulation*. Len Mor.

This memorandum outlines the specifications for a problem-oriented language which is being implemented at the Computer Centre. The language enables a user to simulate dynamic systems on a digital computer using analogue computing techniques.

TECHNICAL MANUAL NO, 3: *WISP*. J.S. Williams.

This manual describes the use of the list processing language WISP, on the GE 225 Computer at the Computer Centre.

PERMUTED INDEX PROGRAM

The GE 225 Manual describing the Permuted Index Program (K2.000) has proved to be somewhat misleading on certain points, as well as containing several errors (see Table 1). There are, however, two known methods of simplifying the program to improve its efficiency.

MISLEADING POINTS

- (a) Throughout the manual, reference is made to "uniquely numbered" or "uniquely identified" lines of text. In fact, the only test made is a comparison of the line number on a card with that on the previous card, in order to determine the continuation of a line of text from one card to the next. If the numbers differ in any way, (i.e. leading zeros, spaces or position of number in the first six columns) the package assumes a new entry. No test is made to see if a new number has been used previously, or if there are leading or trailing zeros or blanks. Thus the following identification numbers are all considered different:

1
000001
1
100000

The last three characters must not be all blanks as this condition is used to signify the end of the list.

Great care must be taken in the preparation of data cards and in the assembly of the final deck, for, if a card is mis-punched or mis-placed, a new line of text will be entered rather than a continuation of the current entry.

- (b) The label written on the output tape is:

BTLOOLSWIC TAPEDATE

This is not the label listed in the manual.

The manual implies that "TAPEDATE" is the date of compilation of the tape, but no facilities are available for placing a date on the tape. The word "TAPEDATE" is written on the tape as part of the label, and no other label can be written.

HELPFUL SUGGESTIONS

- (a) Discard the sort package provided and compile a separate sort program using the FORWARD Sort/Merge Generator. This makes the program more flexible as the editing facility of the sort program can be used to improve the output appearance. In addition, the supplied sort program has not been run satisfactorily and therefore cannot be guaranteed.
- (b) Unless Part III of the dictionary is absolutely necessary, it should be modified or discarded. A test run will show those words required in the dictionary and these may be included with Part II, which is produced by the user. Each entry in the exclusion dictionary must be scanned with each output record, and thus there is an advantage in reducing the size of the dictionary.

Table 1. Errors in Permuted Index Program Manual

Page	Location	Error
1	Paragraph 7	The word "uniquely" may be deleted
1	Paragraph 9	The word "uniquely" may be deleted
13	Paragraph B	The word "different" in lines 5 and 6 should be deleted
15	Program Card	Col. 31 - 39 should be "SWIC TAPE DATE"
15	Input Card	Col. 31 - 45 should be "SWIC TAPE DATE"
15	Output Card	Col. 31 - 45 should be "SWIC TAPE DATE"
19	Paragraph 2	The word "uniquely" in line 3 may be deleted
20	Paragraph 1, Cols. 1 - 6	The statement "all identifying combinations are right-justified with leading zeros" is not true
20	Paragraph 1, Cols. 7 - 9	The words "and are not transmitted with the line of text" may be added
C12	Fig. 9	Constant C140 OCT 61 should read C140 OCT 140

CONCLUSION

A modified version of this program is being prepared at the Computer Centre and shortly will be released.

WISP - A LIST PROCESSING LANGUAGE

by W.N. Fulton,
J.S. Williams

APPLICATIONS OF LIST PROCESSING

List processing languages provide a convenient means of manipulating non-numeric data, the length and structure of which may vary greatly during the solution of a problem.

List processing techniques have proved valuable in:

1. construction of compilers;
2. generation and verification of mathematical proofs;
3. pattern recognition;
4. algebraic manipulation;
5. information retrieval;
6. heuristic programming;
7. linguistic analysis.

THE WISP LANGUAGE

Data storage in WISP is accomplished by means of a *list structure*, which is made up of *list elements*. Each list element may be thought of as a box with two compartments, the left one called the *CAR* and the right one called the *CDR*, either of which may contain an *atom* or a *pointer*. An atom is any legal character, and a pointer is the address of a list element. A *linear list* is a set of list elements in which the CDR of the first contains a pointer to the second, the CDR of the second points to the third, etc. (Figure 2). The CAR of every element of a linear list must contain an atom. A *branched list* is a list in which the CAR of some element contains a pointer, as in Figure 5.

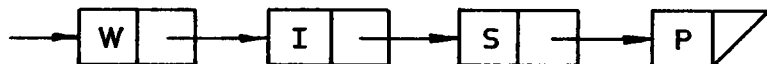



Figure 2. A Linear List

ATOMS, LIST NAMES, AND PUNCTUATION

Each atom corresponds to a special list element, called a *base register*, which occupies a fixed memory location. Each atom may be used as a list name, and the CDR of the atom's base register points to the first element of the list named. In Figure 4, for example, lists A and P are the same (i.e. base registers A and P point to the same element) and contain the characters "W O O D". Base register elements are not generally shown when the list is diagrammed, but the list name is given to indicate the element to which the base register is pointing. A special atom, *NIL*, is used to indicate an empty CAR or CDR, and is represented diagrammatically as .

In the text of a WISP program, a quote (') is used to distinguish between the character standing for itself (i.e. an atom) and a character standing for the corresponding list. Thus 'A denotes the atom A, while A denotes the list A. CAR A denotes the contents of the CAR of the element pointed to by A and CDR A, the CDR of the same element.

ASSIGNMENT STATEMENTS

Initially, each base register points to a single, empty element, i.e. one whose CAR and CDR both contain NIL. The remaining elements are built into a linear list (called the *free list*) which is not directly accessible to the programmer. Additional elements can be obtained from the free list by the use of any one of the following three statements:

```
* = CDR *.
CAR * = CDR *.
CDR * = CDR *.
```

In this example, an asterisk is used as a generic form for any list name. If any one of these is executed when the reference CDR contains NIL, a new element will be taken from the free list and its address placed in the CDR *before* the contents of the CDR are copied (see Figure 3).

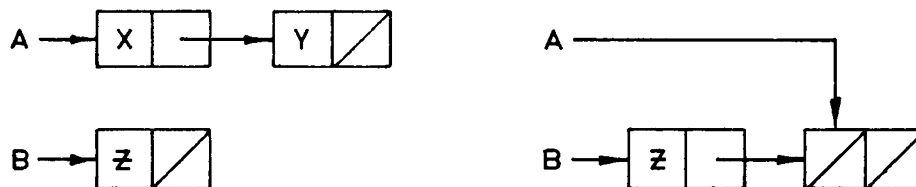


Figure 3. A = CDR B

If the referenced CDR does not contain NIL, no element is added.

As an example of the use of assignment statements, consider the basic operations of building and altering a list. The creation of a list, B, which contains the word "COW" is shown in the following example:

```

A = B,
CAR A = 'C, A = CDR A.
CAR A = 'O, A = CDR A.
CAR A = 'W.

```

Example 2. Creating a List Containing the Characters C, O, and W

This simple sequence illustrates two important concepts:

- (a) the use of a *travelling pointer*, and
- (b) the obtaining of elements from the free list.

The word "COW" was to be stored on list B. Since the base register B must always point to the first element of the list, a second base register is used to point to elements further down the list. In this program, A was chosen to be the travelling pointer. Assuming that list B was originally in its initial state (i.e. with one empty element attached to the base register), successive characters required the addition of elements from the free list. This was accomplished by the use of "A = CDR A,".

A new element may be inserted to alter list B to contain the word "CROW", as follows:

```

CDR C = CDR B, CDR B = C.

```

This element is then filled by:

```

CAR C = 'R

```

As a final example, suppose that we wish to transform the list "CROW" into "ROW". This is done simply by:

```

B = CDR B.

```

Note that no element is obtained from the free list, because CDR B is not empty.

It is significant to point out that the original first element of B (i.e. that which contains the character "C") is now inaccessible to the programmer. There is no way of "backing up" along a list.

CONTROL STATEMENTS

Any WISP statement may be given a name consisting of any string of

alphabetic or numeric characters followed by a delimiter. Control may be transferred to a named statement by the use of the unconditional transfer "TO ** ", where ** is a generic term for any name.

WISP also provides conditional transfers, of the form:

```
TO ** IF (condition)
```

Control continues in sequence except that a transfer of control to the indicated statement will be made when the condition is true. Most conditions resemble assignment statements, and should be self-explanatory. The condition "CAR * = ATOM" is true when the referenced CAR either contains a normal atom or is empty (i.e. contains NIL).

To illustrate the use of these operations, consider a routine to change all of the commas on a linear list B to periods, as shown in Example 3. Here the name "START" refers to the statement "A = B", while "LOOP" refers to "TO FIN IF CAR A \neq ', " and FIN refers to "TO OUT IF CDR A = NIL" (i.e. end of list A). The routine is entered by transferring to "START" and goes to "OUT" (not shown) when the entire list B has been scanned.

```
START, a = b.  
LOOP, TO FIN IF CAR A  $\neq$  ',, CAR A = '..  
FIN.  
TO OUT IF CDR A = NIL, A = CDR A, TO LOOP.
```

Example 3. Changing Commas to Periods

TECHNIQUES OF LIST PROCESSING

Character Pattern Recognition

Suppose that the word "TIMBER" is to replace the word "WOOD" each time it appears in some text. These strings of characters are stored in lists B and A respectively (see Figure 4). The blank at the end of each list is regarded as part of the character string to avoid changing a word such as WOODEN into TIMBEREN.

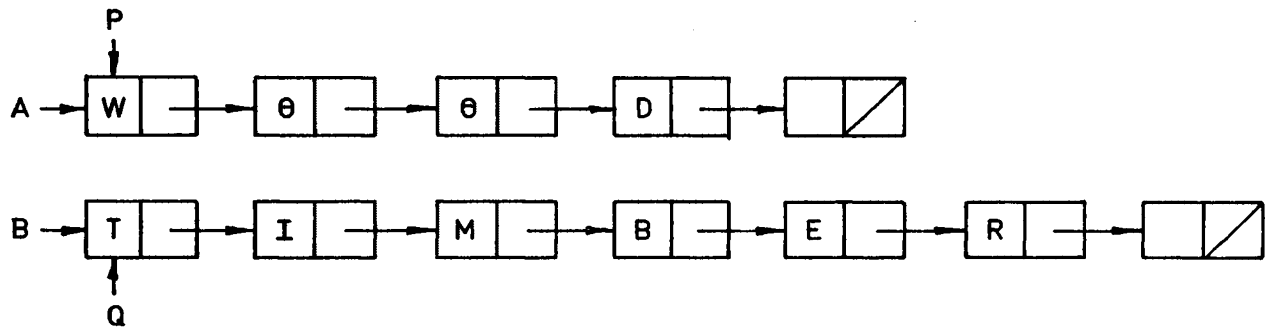


Figure 4. List B Characters to Replace List A Characters in Text

P and Q have been chosen to act as *travelling pointers*.

If text is in list C and a pointer R is used to scan the list, characters can be compared by using a statement such as:

"TO EQUAL IF CAR P = CAR R."

where EQUAL is a statement label.

The end of a list is detected by testing for NIL in a CDR.

Tree Structure

The Tree Structure is very important in list processing and is the basis for programs in such fields as information retrieval, language translation and compiler writing. This structure applies to the situation where a large number of character strings exists, each string possessing its own associated information.

Suppose it is desired to be able to find some information (e.g. age and sex) relating to a specific person from a large number of names. Consider Table 2

Table 2. Names and Associated Information

Name	Associated Information	
	Age	Sex
SMITH	24	M
WALSH	41	M
WALTON	63	F
WILCOX	19	F

These names could be stored in a tree structure together with the associated information. Such a tree is shown in Figure 5 where a period is used to separate the name from the information.

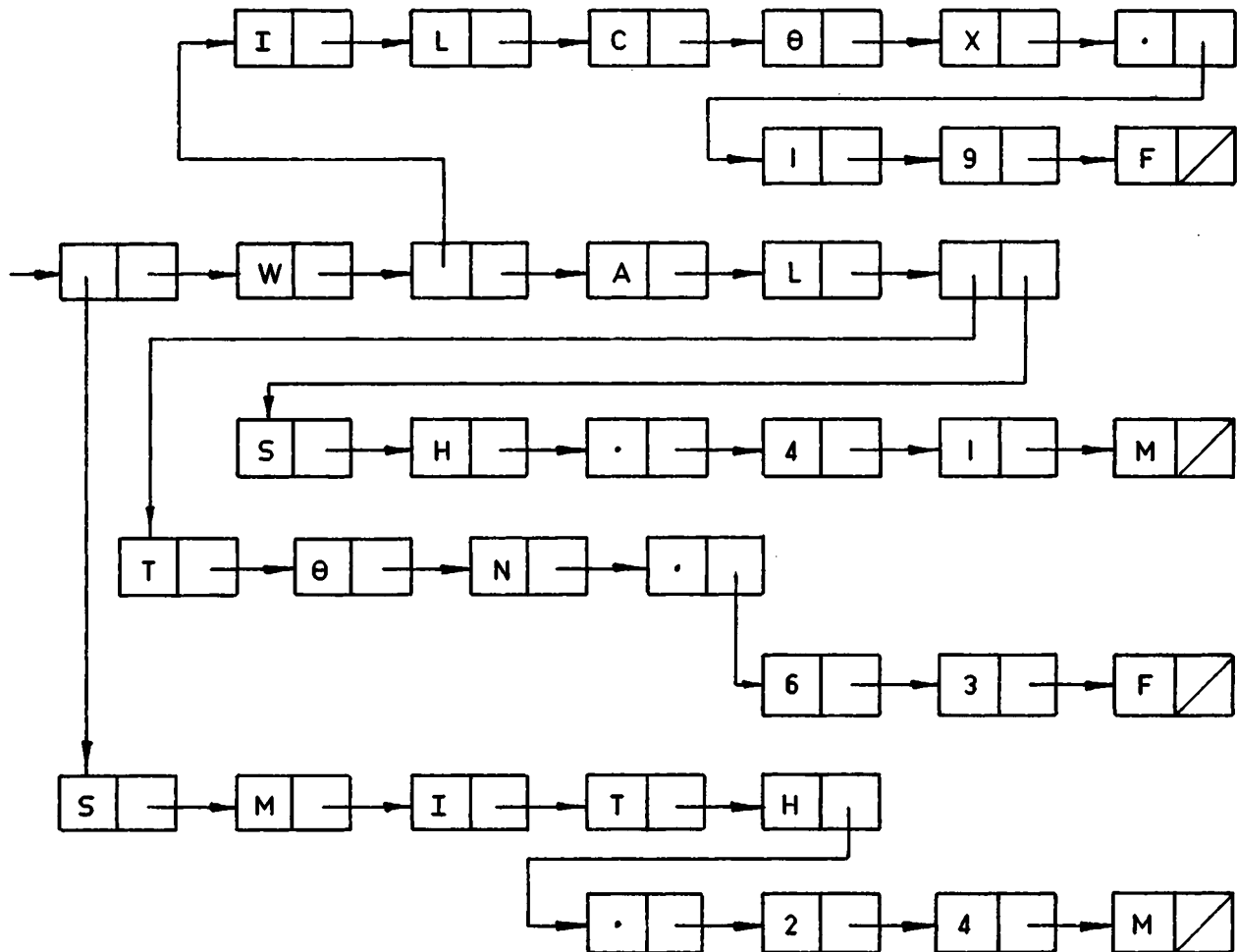


Figure 5. Tree Structure

Figure 5 shows that all names start from a common point. By matching characters with characters in the desired name and choosing the correct path at a branch, the desired information can easily be found.

Symbol Manipulation

An example of symbol manipulation is the process of symbolic differentiation. Suppose the following expression is to be differentiated.

$$y = \frac{(x + 3)(x - 1)}{x^2 - 4}$$

The human approach is to note that this is of the form $\frac{u}{v}$ and so

$$\frac{dy}{dx} = \frac{v \frac{du}{dx} - u \frac{dv}{dx}}{v^2}$$

Then it is seen that u is a product and the appropriate rule of differentiating is used to obtain $\frac{du}{dx}$. The use of a list structure such as that shown in Figure 6 allows these rules to be applied in a convenient manner.

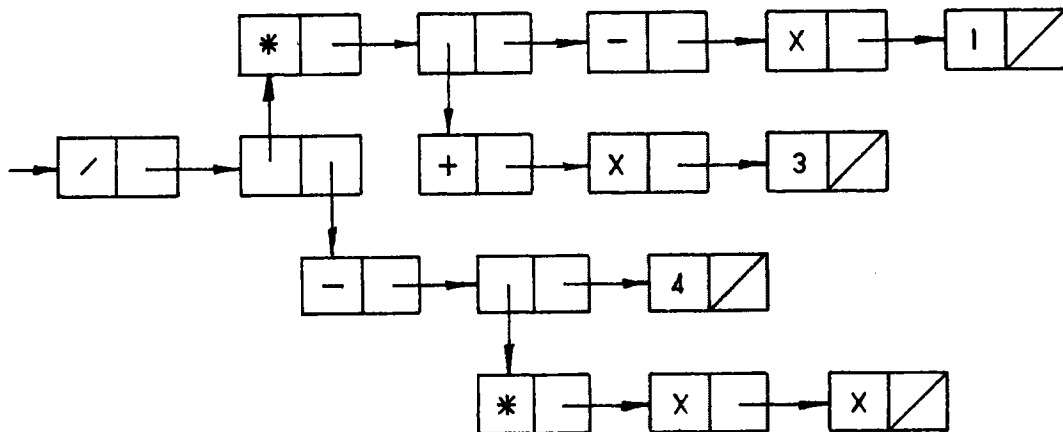


Figure 6. Convenient Symbol Manipulation

Network Structure

A list structure can be created to represent a network such as the network in Figure 7. A sublist can be associated with each component of the network (e.g. component A.) In this way, the first element in each sublist has its CDR pointing to an element which contains the component to which the sublist corresponds. Similarly, the CAR of the first element points to a list consisting of elements that contain the components joined to this particular component. (See Figure 8.) A heuristic approach, in a problem such as circuit layout, can be used to manipulate the sublists and, in this way, re-arrange the components to comply with pre-arranged rules.

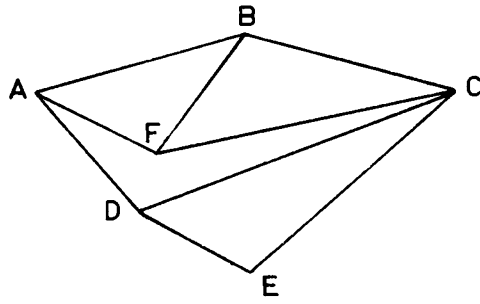


Figure 7. Simple Network

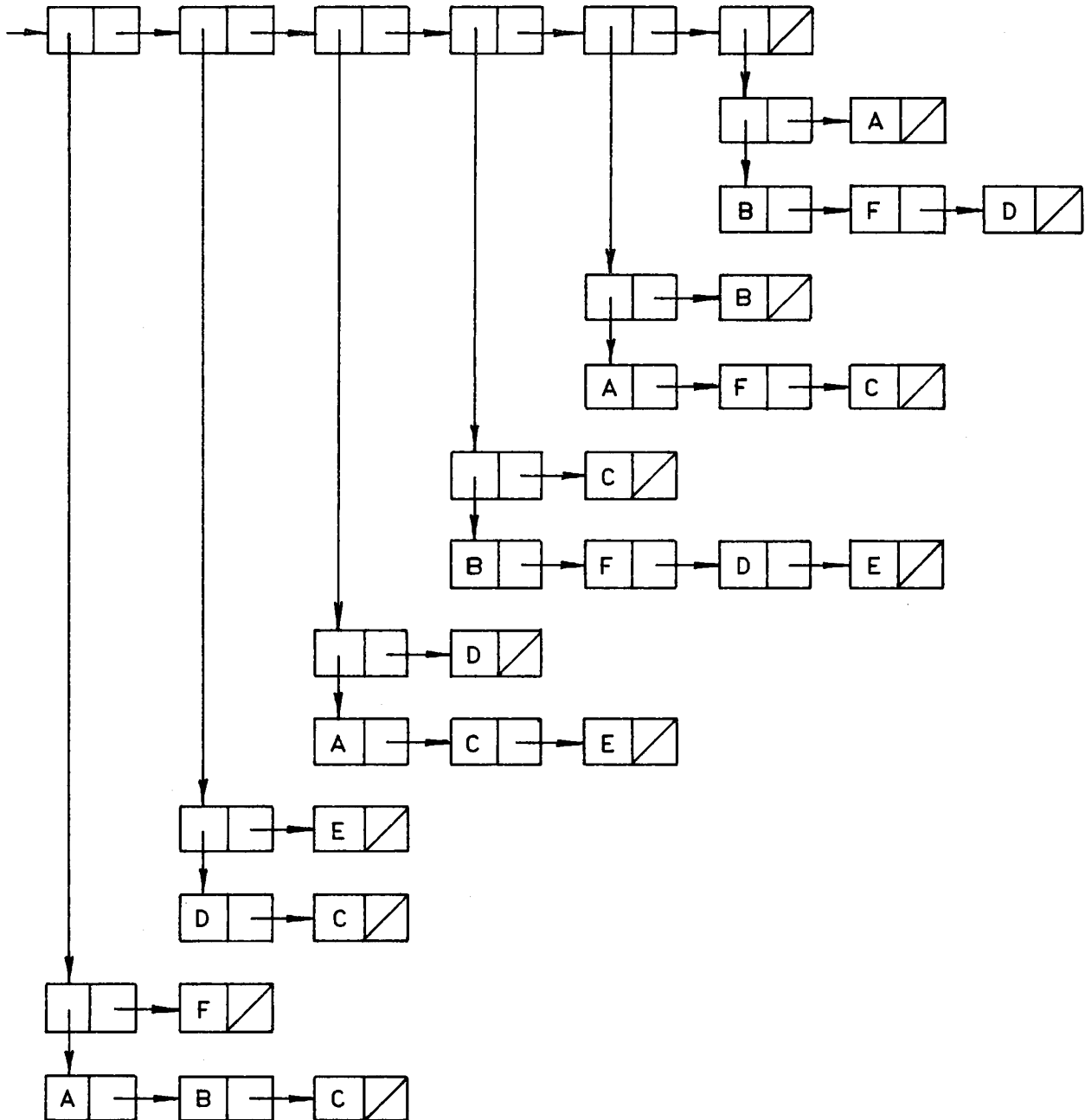


Figure 8. Network Structure List

MAILING LIST

The Bulletin is distributed according to two mailing lists, one for organizations and departments and the other for individuals. If you wish to receive copies of the Bulletin, please complete the appropriate section of the following form, sign and return it to:

The Editor,
Computer Centre Bulletin,
Computer Centre,
University of Queensland,
St. Lucia.
Queensland 4067.

Application for Individual Circulation	Application for Organizational Circulation
--	--

NAME:	POSITION:
-------------	-----------------

POSITION:	ORGANIZATION:
-----------------	---------------------

ADDRESS:	ADDRESS:
----------------	----------------

.....
-------	-------

.....
-------	-------

NUMBER OF COPIES:

SIGNED:	DATE:
---------------	-------------
